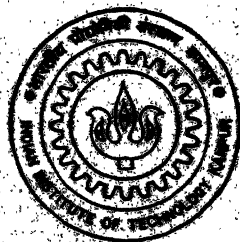


KALKI : **A Human Body Animation System**

by
Kathi Hari Babu



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
FEBRUARY, 1995

KALKI : A Human Body Animation System

*A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of*

MASTER OF TECHNOLOGY

by
Kathi Hari Babu

to the
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR**

Feb, 1995

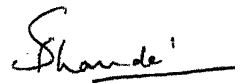
22 MAR 1995/CSE
CENTRE ARY
111 UNFOR

442 No. A. **119118**

CSE-1995-M-BAB-KAL

Certificate

This is to certify that the work contained in this thesis entitled **KALKI : A Human Body Animation System** by **Kathi Hari Babu (Roll No: 9311109)**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.



Dr. Sanjay G. Dhande
Professor,
Department of Computer
Science and Engineering
IIT, Kanpur

March 5, 1995

Abstract

This work aims to build a human body animation system that is easy to use and can produce any type of human motion. The human body is modeled as an articulated tree having 32 degrees of freedom. The system has three modules for designing, animating, and editing sequences of human motion. The user can interactively design a motion, see the resulting animation, and if that is not satisfactory, he can edit the motion until he gets the desired result. A number of tools have been provided to the user for this purpose. These tools in turn use the formulations like forward kinematics, and inverse kinematics from the field of robotics.

The motion of the human body is specified using a so-called *pose file*. These pose files contain the necessary information about the key-frames in the animation. In *Kalki*, we implemented a pose-file hierarchy using which multiple pose-files can be combined. There are three types of pose files in *Kalki*. The so-called *dot-one*, *dot-two* and, *dot-three* pose-files. In the first level we have the dot-one pose file, in which we have the information about the body posture, camera setting, and the number of frames that have to be generated between this and the previous key-frame. At the next level we have dot-two files, in which we have the information about a number of dot-one files, and how many times a dot-one file has to be repeated. When a dot-one file is repeated, we automatically adjust the position of the human body such that there won't be any discontinuity in the animation.

Finally we have dot-three files, where we store the information about the different characters involved in the animation, and the corresponding dot-two files. This pose-file hierarchy provides a frame-work in which we can produce complicated animation involving multiple characters.

**dedicated to
my parents**

Acknowledgements

My heartfelt thanks to my supervisor Prof. S.G.Dhande who introduced me to this fascinating field of computer graphics and human body animation. Without his guidance and understanding I could not have completed this. I am indebted to the robotics textbook by Fu, Gongalez, and Lee from which I freely borrowed the material¹. I would like to thank the staff at the CAD center, in particular P.V.M.Rao, for their help and company. My thanks to all my batch mates who made the life at I.I.T. Kanpur quite enjoyable despite my frequent attacks of asthma. My special thanks to B.Muralidhar for his invaluable friendship and unforgettable company. Finally I would like to thank the authors of \TeX and \LaTeX using which this document is typeset.

¹Lesser artists borrow, great artists steal.

Contents

1	Introduction	1
1.1	What's it all about?	1
1.2	Motion Control and Specification	2
1.3	Literature Review	3
1.4	About <i>Kalki</i>	4
1.5	Organization of Thesis	5
2	Forward Kinematics	6
2.1	Introduction	6
2.2	The Link and Joint Parameters	7
2.3	The Denavit-Hartenberg Representation	8
2.4	Link Coordinate System Assignment	11
2.5	Kinematic equations for manipulators	12
2.6	Articulated chains in <i>Kalki</i>	13
3	Inverse Kinematics	22
3.1	Introduction	22
3.2	Uicker's Iterative Method	22
3.2.1	Matrix equation of approximation	23
3.2.2	Derivation of simultaneous linear equations	25
3.2.3	Solving the simultaneous equations	28
3.3	Inverse kinematics in <i>Kalki</i>	29

4	System Design	30
4.1	Introduction	30
4.2	Pose file hierarchy	31
4.3	Pose file design	32
4.4	Pose file editing	32
4.5	Pose file animation	32
4.6	Stability checking	33
4.7	Scope for extensions	33
4.8	Starbase implementation	34
5	Using <i>Kalki</i>	36
5.1	Introduction	36
5.2	Design a dot-1 file	37
5.2.1	Changing the cameras	39
5.2.2	Change Kalki's pose	39
5.2.3	Save the Pose	40
5.2.4	Load a pose	40
5.2.5	Initialize kalki	41
5.2.6	Use inverse kinematics	41
5.2.7	Fix the limbs	43
5.3	Design a Dot-2 pose file	43
5.4	Design a dot-3 pose file	44
5.5	Animate a dot-3 pose file	44
5.6	Edit a dot-1 pose file	47
5.6.1	Edit a new dot-1 file	47
5.6.2	Translate poses	48
5.6.3	Translate cameras	48
5.6.4	Edit a record.	49
6	Conclusions	50
	Bibliography	52

List of Figures

2.1	The forward kinematics problem	7
2.2	Link coordinate system and it's parameters	9
2.3	Articulation points and degrees of freedom in <i>Kalki</i>	15
2.4	Coordinate frame assignment form base to pelvis	16
2.5	Coordinate frame assignment form pelvis to head	17
2.6	Coordinate frame assignment form pelvis to foot	19
2.7	Coordinate frame assignment form shoulder to palm	21
3.1	The inverse kinematics problem	23
4.1	System structure in <i>Kalki</i>	31
5.1	Main menu in <i>Kalki</i>	37
5.2	Changing body posture in <i>Kalki</i>	40
5.3	Specifying camera for dot-1 file.	40
5.4	Using inverse kinematics in <i>Kalki</i>	42
5.5	Doing pushups in <i>Kalki</i>	44
5.6	Multiple character animation in <i>Kalki</i>	45
5.7	Dancing in progress.	45
5.8	Editing a pose in <i>Kalki</i>	47

List of Tables

2.1	Joint and link parameters for base to pelvis articulated chain	16
2.2	Joint and link parameters for pelvis to head articulated chain	17
2.3	Joint and link parameters for pelvis to foot articulated chain	20
2.4	Joint and link parameters for shoulder to palm articulated chain	21

Chapter 1

Introduction

Should we force science down the throats of those that have no taste for it? Is it our duty to drag them kicking and screaming into the twenty-first century? I am afraid that it is.

George Porter, British chemist.

1.1 What's it all about?

Realistic animation of human body is an interesting and important problem. This has applications in a variety of fields. Computer graphics and human body animation have already made their debut in movies, advertisements, and other entertainment media. Besides this, as the technology is advancing, we find more and more situations where we need to simulate human behaviour. Astronauts in space, movement of people in busy traffic, reaction of a car driver during a car accident are some of the examples. One other interesting application is the study of ergonomics. This is used to find the amount of energy a man spends during a particular movement. So, using graphical simulations we can design the furniture and the environment such that the workman's fatigue is reduced.

Interesting as it is, human body animation poses several problems. In the first place, this is a very complex modeling problem. The human body is generally modeled as an articulated chain made up of different limbs. The motion of the limbs relative to each other is somewhat restricted. Also the number of total degrees of freedom is quite high. To solve these problems we apply the techniques developed in the field of robotics.

There is also a trade-off between the level of specification and the amount of control. In the simplest case, where we specify all the degrees of freedom at each frame, we have total control over the animation. But this means a huge amount of input to the system. For real

time animation we have to generate about 30 frames per second. So, to generate t seconds of animation for an n degrees of freedom object, we have to input $30nt$ numbers. A one minute animation for an object with 50 degrees of freedom will require 90,000 numbers to completely specify the motion.

This forces us to consider higher levels of specification. Animation languages, command libraries, finite state machines, inverse kinematics, and dynamics are among the many methods explored. As the level of specification gets higher, we start losing the control and are required to know more about computers and the underlying mathematics. Such a situation is clearly undesirable, since we expect these systems to be used by animators who have little knowledge in computer science.

To add on to the difficulties is the expectations of people. Since we are very intimately acquainted with the movements of the human body, any slight disparity will be quickly detected. People would accept the motion of a flying saucer any way we show it to them, but they are generally finicky in the case of a human body.

1.2 Motion Control and Specification

Motion control systems are often classified as interactive, scripted or a combination of both. Interactive implies that the user and the motion control system participate in a loop where the user describes a motion, the computer quickly provides an animation using it, the user modifies the motion as necessary, etc. In scripted methods, the user creates a written script describing the motion that the control system later interprets to produce the animation.

Motion control can also be classified as low level, high level, or somewhere in between. Low level suggests that the user is required to specifically describe the motion for individual degrees of freedom, such as a path through space or motion at joints. With high level control, the user may describe motion in more general terms, as in “walk forward”, leaving the system to find the appropriate low level motion description. High level control is desirable, because the user can succinctly and quickly define complex motion. It does have some disadvantages, however, because the user has less control over the exact motion. For this reason, systems typically provide a combination of high-level and low-level techniques.

Objects vary in the types of motion available to them. Typically motion is described in terms of degrees of freedom, that is the number of independent coordinates needed to specify

the positions of all components in the system. The following types of objects commonly appear in practice:

Particles. A particle can be described as a point in three dimensional space. The location and motion of such a point is designated by three variables. Animating a point requires a triplet of numbers for each animation frame, or three functions describing the variations of x , y and z over time.

Rigid bodies. A rigid body is defined by some number of points that must move together. They may not move relative to each other. The motion of a rigid body is specified by six degrees of freedom. Three degrees for specifying translation, and three for specifying orientation.

Flexible bodies. A flexible body consists of an infinite number of points which move relative to each other over time. In practice a flexible body may be defined as a set of moving points. These points generally represent control points for surfaces. A flexible body having p control points has $(3 \times p)$ degrees of freedom varying over time.

Articulated bodies. Articulated bodies are made up of segments whose motion relative to each other is somewhat restricted. For example, a human body is often represented as rigid segments joined at articulations (joints) which have one to three degrees of freedom. The total degrees of freedom that must be specified is the sum of the number of degrees of freedom at each joint.

1.3 Literature Review

A project like this involves literature survey in a variety of fields. Most of the mathematical preliminaries are covered either in the books of graphics or in the books of robotics. The graphics text book by Foely and Vandam gives a detailed account of the transformation matrices which are the essential tools in any graphical modeling. This topic is also covered in the robotics text book by Fu, Gonzalez, and Lee. Human body animation has a rich source of publications in all the major graphics journals. This field has attracted the interest of the graphics people, robotics people, and biomechanics people. Especially the two journals "ACM SIGGRAPH" and "IEEE Transactions on Computer Graphics and Applications" provide a comprehensive study of this field.

The earliest papers dealing with the formalisms in robotics appeared in the 1950's. Later on many papers appeared in the fields of inverse kinematics and robot arm dynamics. The "International journal of robotics research" contains many papers that reflect the state of the art research in robotics. Initially people used ad-hoc methods for representing the human body. After that the *Denavit-Hartenberg* representation has become a standard for representing the human body as an articulated chain.

M.Girard and others [12] used forward kinematics and inverse kinematics for generating human motion. Their implementation, however, concentrates only on human walking. The graphics community quickly realized that to generate arbitrary sequences of human motion, we should have the ability to position the human body as we wish with ease. Badler and others [3] [5] worked in this regard. Later on the emphasis shifted towards the application forces and torques. Use of forward dynamics has been tried. [8] [6] are only some of the many papers appeared in this field.

1.4 About *Kalki*

Kalki is a human body animation system designed to meet the following goals.

- Sufficiently higher level of specification.
- Full control over the animation.
- Ease of use for an animator who is not familiar with computer science.
- Ease of enhancing functionality or efficiency.

Most systems designed for human body animation hard wire specific motor skills into the system. As a result it becomes very difficult to achieve motion other than that programmed in to it. For example, if we have a system designed for human walking, we can't use it to generate dancing or gymnastics. Keeping this in view, *Kalki* is not hard wired for any particular motion. *Kalki* is very modular and open ended. Further modules enhancing it's operation can be added with minimum difficulty.

In *Kalki* the user can interactively design sequences of human motion. He can see a test run and if he doesn't like the look of it, he can edit the key-frames and adjust the time distances between the key-frames. He is provided with a variety of tools that facilitate the

design and editing of the key-frames. These tools in turn use the basic formalisms developed in robotics.

1.5 Organization of Thesis

The rest of the report is organized as follows. First we discuss the basic mathematical formalisms in robotics. In chapter-2 forward kinematics and the issue of consistently representing an articulated body is considered. This chapter also contains the figures and tables for the different articulated chains present in the human body. Chapter-3 discusses in detail the Uicker's iterative method of inverse kinematics. The treatment here differs from the original paper by Uicker in two aspects. We have modified the notations so as to make it consistent with the earlier notations. We removed two restrictions that are present in the original Uicker's paper.

Next in Chapter-4 the issues of stability and dynamics are addressed. The problems that arise if we want to implement stable motion, and the the stability algorithm that has been implemented in *Kalki* are given. The *Lagrangian-Euler* formulation for robot arm forward dynamics is considered in this chapter. Though there are other dynamics formulations which are faster than this, this is chosen because it is very much consistent with the notations developed in the previous chapters. So long as we don't need a real-time dynamics routine, this will serve our purpose.

Chapter-5 deals with the current implementation. It outlines the underlying philosophy of *Kalki* and it's powerful pose file hierarchy. This also gives the suitable extensions and the issues related to the Starbase implementation on the HP-TSRX work-stations. Chapter-6 concludes the report with an overview of human body animation and the current research trends in this field.

Chapter 2

Forward Kinematics

From an evolutionary point of view, man has stopped moving, if he ever did move.

Pierre Teilhard de Chardin, French palaeontologist.

2.1 Introduction

In this Chapter we shall consider the basic modeling of the human body. In this and the subsequent chapters, we will be dealing with many terms related to robotics. So, before we start let us understand their definitions.

Link: A rigid body that has a position and orientation. In *Kalki* all the limbs are modeled as rigid bodies. So, thigh, neck, etc... are all links.

Joint: This connects two links and defines the type of motion relative to each other. For example a revolute joint enables it's two links to rotate with respect to each other.

Articulated Chain: This is a chain of links joined in series by joints. There are five *articulated chains* in the human body implemented. They are

1. Pelvis to left foot
2. Pelvis to right foot
3. Pelvis to head
4. Left shoulder to left palm
5. Right shoulder to right palm

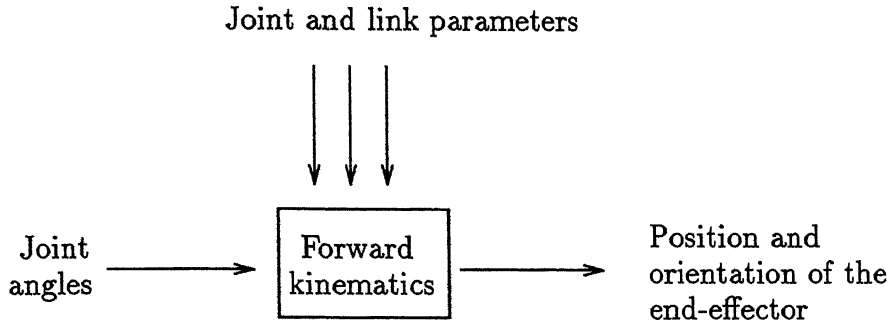


Figure 2.1: The forward kinematics problem

Base: This is the first link in an articulated chain. For example pelvis is the base for the first three articulated chains mentioned above.

End-effector: This is the last link in an articulated chain. Head, left foot, right foot, left palm, right palm are the different *end-effectors* in the human body.

An articulated chain has some parameters which describe its geometric structure. Some of these parameters are fixed (like the lengths of the links). Some are variable (like the angles at the joints). In robotics, the field of kinematics deals with the study of the geometry of the robot arm as the variable parameters change with time. In particular the forward kinematics problem or the direct kinematics problem addresses the issue of finding the position and orientation of the end-effector with respect to a fixed coordinate system as the joint angles change. Formally stating we want to find out

For a given manipulator, given the joint angle vector $\mathbf{q}(t) = (q_1(t), q_2(t), \dots, q_n(t))^T$ and the geometric link parameters, where n is the number of degrees of freedom, what is the position and orientation of the end-effector of the manipulator with respect to a reference coordinate system?

2.2 The Link and Joint Parameters

A mechanical manipulator consists of a sequence of rigid bodies, called links, connected by revolute or prismatic (which we will not consider) joints. Each joint-link pair constitutes 1 degree of freedom. Hence for an N degree of freedom manipulator, there are N joint-link pairs with link 0 (not considered part of the robot) attached to a supporting base where an inertial coordinate system is usually established for this dynamic system and the last link is

attached to the tool. The joints and links are numbered outwardly from the base: thus, joint 1 is the point of connection between link 1 and the supporting base. Each link is connected to at most two others so that no closed loops are formed. A joint axis is established at the connection of two links. This joint axis will have two normals connected to it, one for each of the links. The relative position of two such connected links (link $(i - 1)$ and link i) is given by d_i which is measured along the joint axis between the two normals. The joint angle θ_i between the normals is measured in a plane normal to the joint axis. Hence, d_i and θ_i may be called the *distance* and the *angle* between the adjacent links, respectively. They determine the relative position of the adjacent links.

A link i is connected to at most two other links (e.g., link $(i - 1)$ and link $(i + 1)$); thus, two joint axes are established at both ends of the connection. The significance of links, from a kinematic perspective, is that they define a fixed configuration between their joints which can be characterized by two parameters: a_i and α_i . The parameter a_i is the shortest distance measured along the common normal between the joint axis (i.e., the z_{i-1} and z_i axis for joint i and joint $(i + 1)$, respectively), and α_i is the angle between the joint axis measured in a plane perpendicular to a_i . Thus a_i and α_i may be called the *length* and the *twist angle* of the link i , respectively. They determine the structure of link i .

In summary four parameters a_i , α_i , d_i , and θ_i , are associated with each link of a manipulator. If a sign convention for each of these parameters has been established, then these parameters constitute a sufficient set to completely determine the kinematic configuration of each link of a robot arm. Note that these four parameters come in pairs: the link parameters (a_i, α_i) which determine the structure of the link and the joint parameters (d_i, θ_i) which determine the relative position of neighboring links.

2.3 The Denavit-Hartenberg Representation

To describe the translational and rotational relationships between the adjacent links Denavit and Hartenberg proposed a matrix method of systematically establishing a coordinate system (body-attached frame) to each link of an articulated chain. The Denavit and Hartenberg (D-H) representation results in a 4×4 homogeneous transformation matrix representing each link's coordinate system at the joint with respect to the previous link's coordinate system. Thus, through sequential transformations the end-effector expressed in the "hand

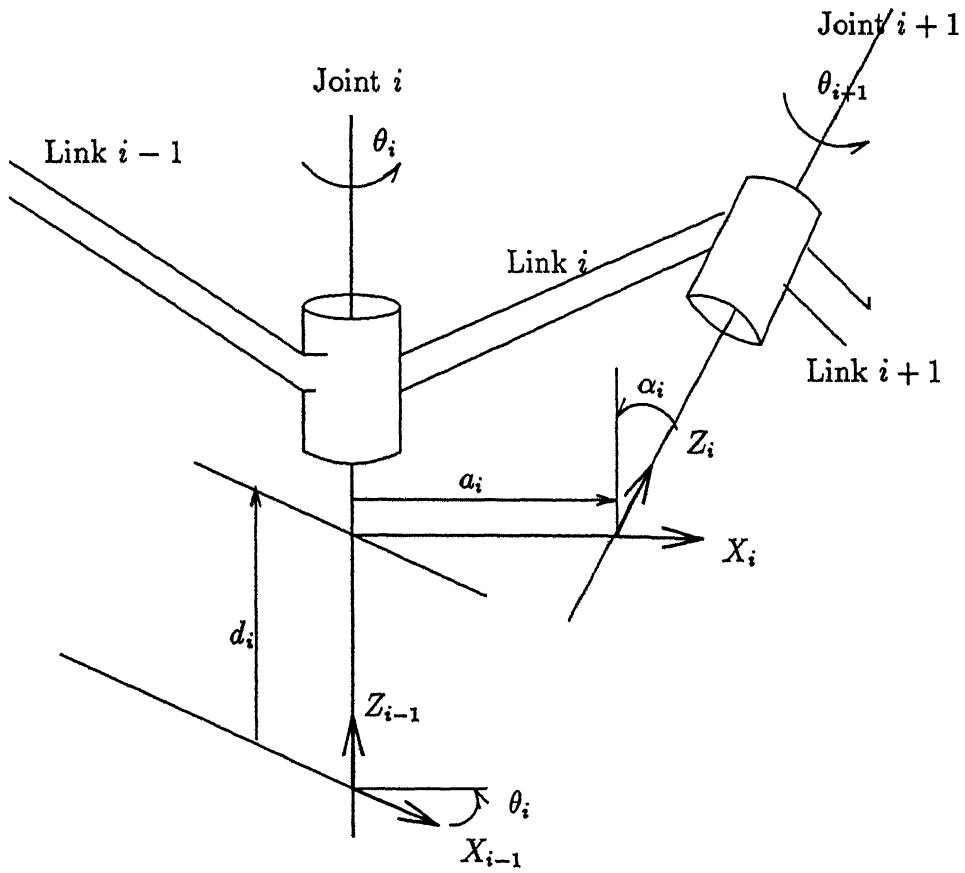


Figure 2.2: Link coordinate system and its parameters

coordinates" can be transformed and expressed in the "base coordinates" which make up the inertial frame of this dynamic system.

An orthonormal coordinate system (x_i, y_i, z_i) can be established for each link at its joint axis, where $i = 1, 2, \dots, n$ (n = number of degrees of freedom) plus the base coordinate frame. Since a rotary joint has only one degree of freedom, each (x_i, y_i, z_i) coordinate frame of a robot arm corresponds to joint $(i + 1)$ and is fixed in link i . Every coordinate frame is determined and established on the basis of three rules:

1. The z_{i-1} axis lies along the axis of motion of the i th joint.
2. The x_i axis is normal to z_{i-1} axis and is pointing away from it.
3. The y_i axis completes the right-handed coordinate system as required.

By these rules, one is free to choose the location of the coordinate frame 0 anywhere in the supporting base, as long as the z_0 axis lies along the motion of the first joint. The last coordinate frame can be placed anywhere in the hand, as long as the x_n axis is normal to the z_{i-1} axis.

The D-H representation of a rigid link depends on four geometric parameters associated with each link. These four parameters completely describe any revolute or prismatic joint. These four parameters are defined as follows:

- θ_i is the joint angle from the x_{i-1} axis to the x_i axis about the z_{i-1} axis (using the right-hand rule)
- d_i is the distance from the origin of the $(i - 1)$ th coordinate frame to the intersection of the z_{i-1} axis with the x_i axis along the z_{i-1} axis.
- a_i is the offset distance from the intersection of the z_{i-1} axis with the x_i axis to the origin of the i th coordinate frame along the x_i axis. (or the shortest distance between the z_{i-1} and the z_i axes)
- α_i is the offset angle from the z_{i-1} axis to the z_i axis about the x_i axis (using the right-hand rule).

For a rotary joint d_i , a_i , and α_i are the joint parameters and remain constant for a robot, while θ_i is the joint variable that changes when link i moves (or rotates) with respect to link $(i - 1)$.

2.4 Link Coordinate System Assignment

With the above three basic rules for establishing an orthogonal coordinate system for each link and geometric interpretation of the joint and link parameters, a procedure for establishing *consistent* orthonormal coordinate systems for a robot is given in the algorithm below.

Given an n degree of freedom robot arm, this algorithm assigns an orthonormal coordinate system to each link of the robot arm according to arm configurations similar to those of human arm geometry. The labeling of the coordinate systems begins from the supporting base to end-effector of the robot arm. The relations between adjacent links can be represented by 4×4 homogeneous transformation matrices. The significance of this procedure is that it will aid the development of a consistent procedure for deriving the joint solution as discussed in later sections¹.

1. **Establish the base coordinate system.** Establish a right-handed orthonormal coordinate system (x_0, y_0, z_0) at the supporting base with the z_0 axis lying along the axis of motion of joint 1 and pointing toward the shoulder of the robot arm. The x_0 and y_0 axes can be conveniently established and are normal to the z_0 axis.
2. **Initialize and loop.** For each i , $i = 1, 2, \dots, n$, perform steps 3 to 6.
3. **Establish joint axis.** Align the z_i with the axis of motion of joint $(i + 1)$. For robots having left-right hand configurations, the z_1 and z_2 axes are pointing away from the shoulder and the “trunk” of the robot arm.
4. **Establish the origin of the i th coordinate system.** Locate the origin of the i th coordinate system at the intersection of the z_i and z_{i-1} axes or at the intersection of the common normal between the z_i and z_{i-1} axes and z_i the axis.
5. **Establish x_i axis.** Establish $x_i = \pm(z_{i-1} \times z_i) / \|z_{i-1} \times z_i\|$ or along the common normal between the z_{i-1} and z_i axes when they are parallel.
6. **Establish y_i axis.** Assign $y_i = \pm(z_i \times x_i) / \|z_i \times x_i\|$ to complete the right-handed coordinate system. (Extend the z_i and x_i axes if necessary for steps 9 to 12)

¹Note that the assignment of coordinate systems is not unique.

7. Establish the hand coordinate system. Establish z_n along the direction of z_{i-1} axis and pointing away from the robot. Establish x_n such that it is normal to both z_n and z_{n-1} axes. Assign y_n to complete the right-handed coordinate system.
8. Find joint and link parameters. For each i , $i = 1, 2, \dots, n$, perform steps 9 to 12.
9. Find d_i . d_i is the distance from the origin of the $(i - 1)$ th coordinate system to the intersection of the z_{i-1} axis and the x_i axis along the z_{i-1} axis.
10. Find a_i . a_i is the distance from the intersection of the z_{i-1} axis and x_i axes to the origin of the i th coordinate system along the x_i axis.
11. Find θ_i . θ_i is the angle of rotation from the x_{i-1} axis to the x_i axis about the z_{i-1} axis. This, generally is the joint variable.
12. Find α_i . α_i is the angle of rotation from the z_{i-1} axis to the z_i axis about the x_i axis.

2.5 Kinematic equations for manipulators

Once the D-H coordinate system has been established for each link, a homogeneous transformation matrix can easily be developed relating the i th coordinate frame to $(i - 1)$ th coordinate frame. From the figure it is obvious that a point r_i expressed in i th coordinate system may be expressed in the $(i - 1)$ th coordinate system as r_{i-1} by performing the following successive transformations:

1. Rotate about the z_{i-1} axis an angle of θ_i to align the x_{i-1} axis with the x_i axis (x_{i-1} axis is parallel to x_i axis and pointing in the same direction).
2. Translate along the z_{i-1} axis a distance of d_i to bring the x_{i-1} and x_i axes in to coincidence.
3. Translate along the x_i axis a distance of a_i to bring the two origins as well as the x axis into coincidence.
4. Rotate about the x_i axis an angle of α_i to bring the two coordinate systems into coincidence.

Each of these four operations can be expressed by a basic homogeneous rotation-translation matrix and the product of these four homogeneous matrices yields a composite homogeneous matrix, ${}^{i-1}A_i$, known as the D-H transformation matrix for adjacent coordinate frames i and $(i-1)$. Thus,

$${}^{i-1}A_i = T_{z,d}T_{z,\theta}T_{x,a}T_{x,\alpha} \quad (2.1)$$

$$= \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.2)$$

Where α_i , a_i , d_i are the constants while θ_i is the joint variable for a revolute joint. Using the ${}^{i-1}A_i$ matrix, one can relate a point p_i at rest in link i , and expressed in homogeneous coordinates with respect to coordinate system i , to the coordinate system $(i-1)$ established at link $(i-1)$ by

$$p_{i-1} = {}^{i-1}A_i p_i \quad (2.3)$$

where $p_{i-1} = (x_{i-1}, y_{i-1}, z_{i-1}, 1)^T$ and $p_i = (x_i, y_i, z_i, 1)^T$. The homogeneous matrix 0T_i which specifies the location of the i th coordinate frame with respect to the base coordinate system is the chain product of successive coordinate transformation matrices of ${}^{i-1}A_i$, and is expressed as

$${}^0T_i = {}^0A_1 {}^1A_2 \dots {}^{i-1}A_i \quad (2.4)$$

2.6 Articulated chains in *Kalki*

In *Kalki* we have three basic articulated chains. The first chain is from the base of the pelvis to the head. The second starts from the pelvis (or the top of the thigh) to the foot. The third one is from the top of the upper arm to the palm. Note that on the whole we have five articulated chains because we have two legs and two palms. Starbase² uses left handed coordinate system. So, before plotting the human body we need to transform the coordinate system at the base of the pelvis to the world coordinate system of Starbase.

²This is the graphics package on the HP-TSRX work stations.

The following are the coordinate system assignments and their corresponding joint and link parameters.

As mentioned earlier, *Kalki* has five articulated chains. First we position and orient the pelvis. The pelvis reference point is fixed at the navel of the human body. Next we draw the pelvis in it's coordinate system. After this, we push a transformation matrix that takes us to the starting of the articulated chain forming the left leg. Similarly we draw the right leg. After this, we pop out the corresponding transformation matrices from the matrix stack. The portion from the pelvis to the upper body is drawn, and again a transformation matrix that takes us to the starting of the left upper arm is pushed onto the transformation matrix stack. Similarly we draw the right hand, and finally we draw the neck and head.

Note: In the following figures, the dotted line indicates that the origin of the first coordinate system coincides with that of the second one. The first coordinate system is drawn separately because that makes the picture clear. In the coordinate frame assignment, the upward slanted lines indicate lines going into the surface of the paper. The downward slanted lines indicate lines coming towards the reader.

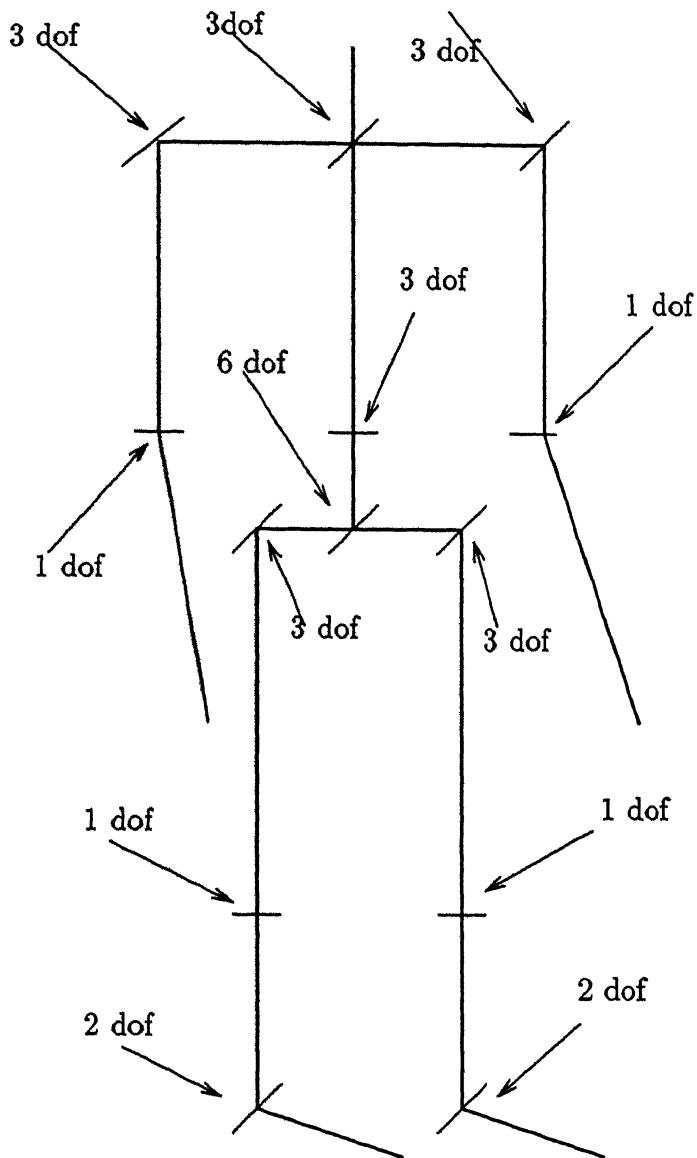


Figure 2.3: Articulation points and degrees of freedom in *Kalki*

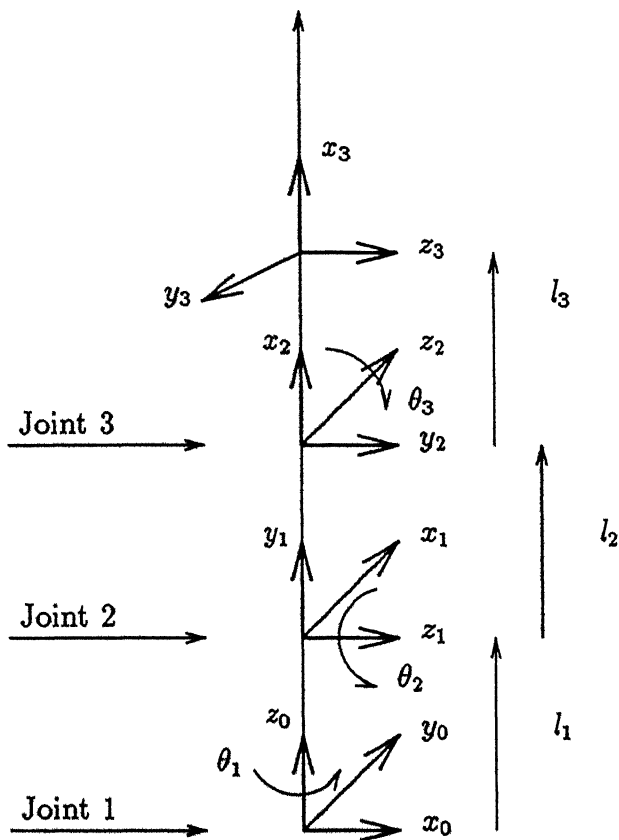


Figure 2.4: Coordinate frame assignment form base to pelvis

a	α	d	θ
0	90 deg	l_1	$90 + \theta_1$ deg
l_2	90 deg	0	$90 + \theta_1$ deg
l_3	90 deg	0	$90 + \theta_1$ deg

Table 2.1: Joint and link parameters for base to pelvis articulated chain

a	α	d	θ
l_1	90 deg	0	$90 + \theta_1$ deg
0	90 deg	0	$90 + \theta_2$ deg
0	90 deg	$l_2 + l_3$	$90 + \theta_3$ deg
l_4	90 deg	0	$90 + \theta_4$ deg
0	90 deg	0	$90 + \theta_5$ deg
0	0 deg	$l_5 + l_6$	$0 + \theta_6$ deg

Table 2.2: Joint and link parameters for pelvis to head articulated chain

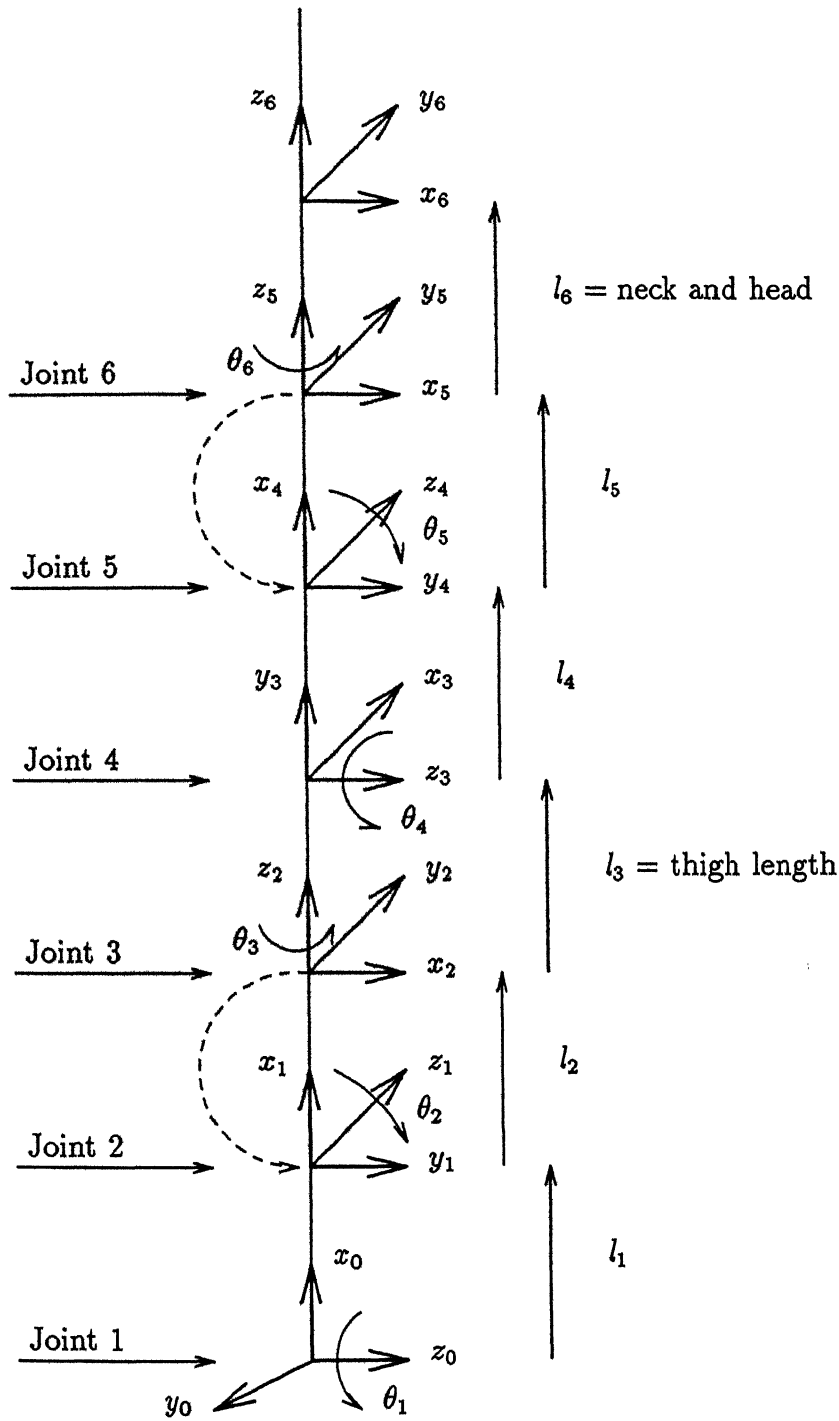


Figure 2.5: Coordinate frame assignment from pelvis to head

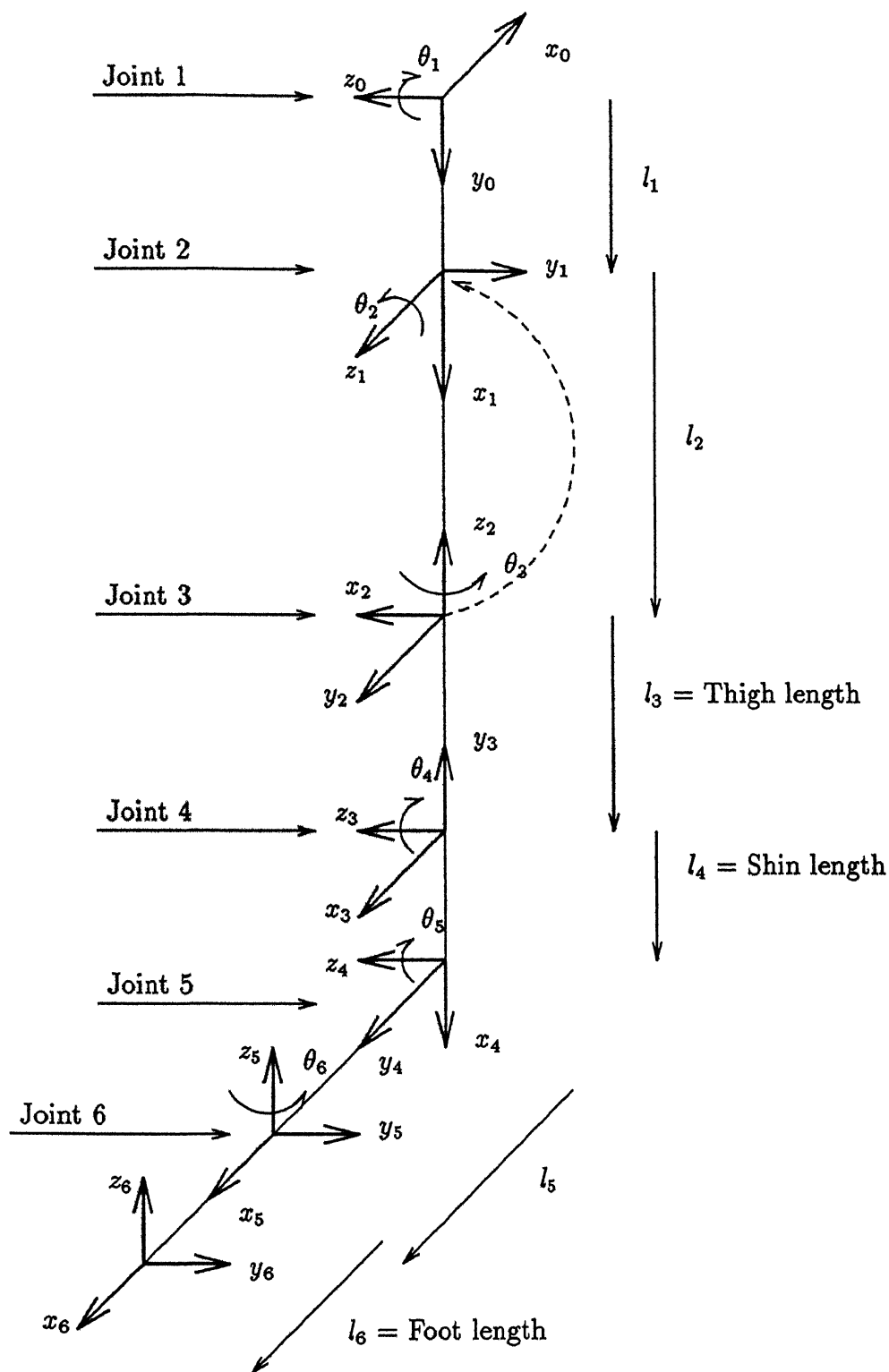


Figure 2.6: Coordinate frame assignment from pelvis to foot

a	α	d	θ
l_1	-90 deg	0	$-90 + \theta_1 \text{ deg}$
0	90 deg	0	$-90 + \theta_2 \text{ deg}$
0	90 deg	$-(l_2 + l_3)$	$90 + \theta_3 \text{ deg}$
l_4	0 deg	0	$-90 + \theta_4 \text{ deg}$
l_5	-90 deg	0	$90 + \theta_5 \text{ deg}$
l_6	0 deg	0	$0 + \theta_6 \text{ deg}$

Table 2.3: Joint and link parameters for pelvis to foot articulated chain

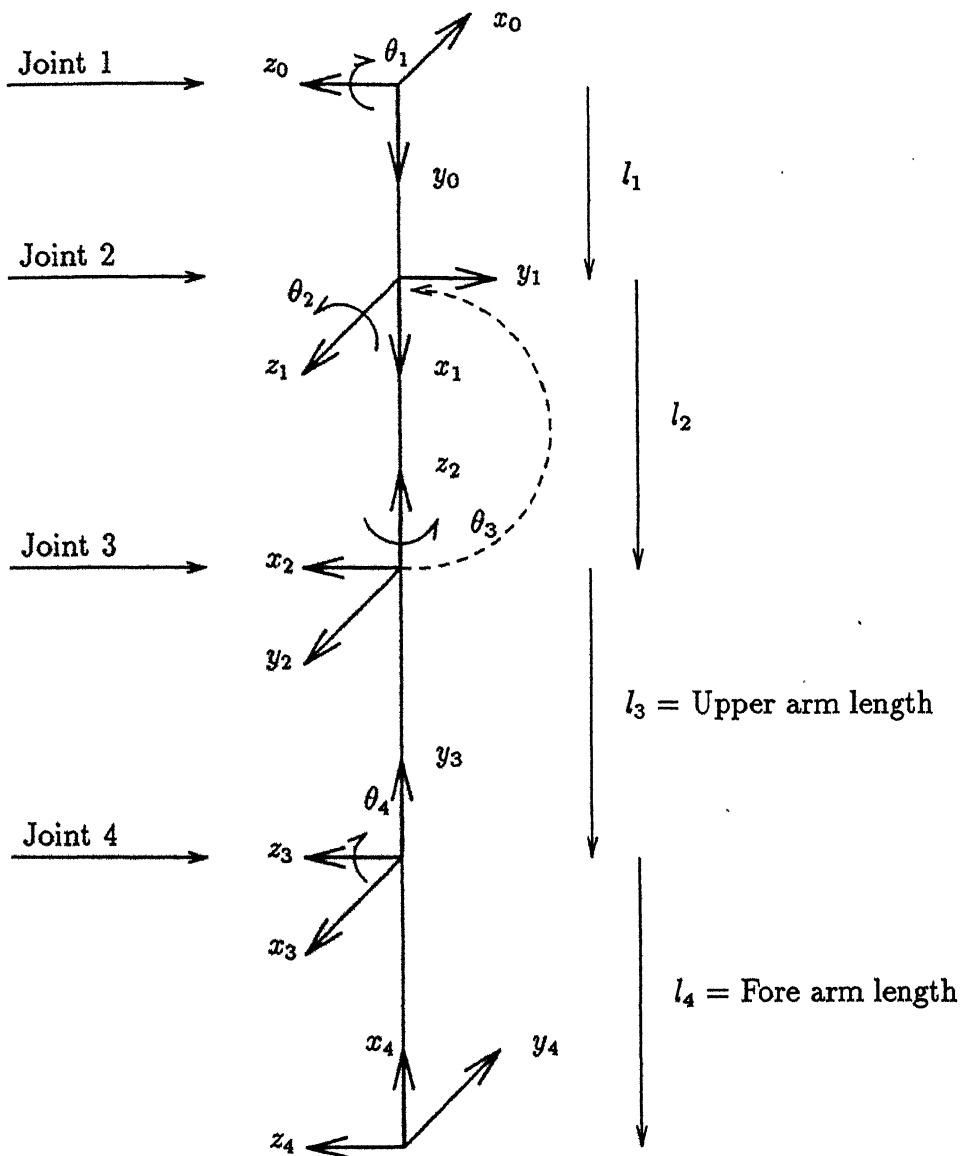


Figure 2.7: Coordinate frame assignment from shoulder to palm

a	α	d	θ
l_1	-90°	0	$90 + \theta_1^\circ$
0	90°	0	$-90 + \theta_2^\circ$
0	90°	$-(l_2 + l_3)$	$90 + \theta_3^\circ$
$-l_4$	0	0	$90 + \theta_4^\circ$

Table 2.4: Joint and link parameters for shoulder to palm articulated chain

CENTRAL LIBRARY
I. I. T., KANPUR
Acc. No. A. 119118

Chapter 3

Inverse Kinematics

First draw your graph, then take your readings.

Murphy's Law.

3.1 Introduction

If we know the the joint angles at each joint, then using forward kinematics we can plot the human body. In designing the key-frames for the human body, many times we only know the positions of certain limbs. For example when a man sits, the position of his feet remain fixed at the same point. In this situation we are confronted with a different problem. The problem here is, given the position and orientation of the end-effector of a robot arm as 0A_n and it's joint and link parameters, we would like to find the corresponding joint angles $\Theta = (\theta_1, \theta_2, \dots, \theta_n)^T$ of the robot so that the end-effector can be positioned as desired. In general the inverse kinematics problem can be solved by various methods, such as inverse transform (Paul et al. [1981]), screw algebra (Kohli and Soni [1975]), dual matrices (Denavit [1956]), dual quaternion (Yang and Freudenstein [1964]), iterative and geometric approaches.

3.2 Uicker's Iterative Method

Uicker proposed an iterative method for the inverse kinematic problem. In his method he considers a simple-closed loop and assumes that the first angle is fixed. We modified this method to remove the above restrictions. In the following method we consider open-loop chains, as are found in the human body, without the restriction that the first angle is fixed. We make use of the D-H notation developed in the previous chapter to solve the inverse kinematic problem. In this method we approximate the joint angles in successive iterations

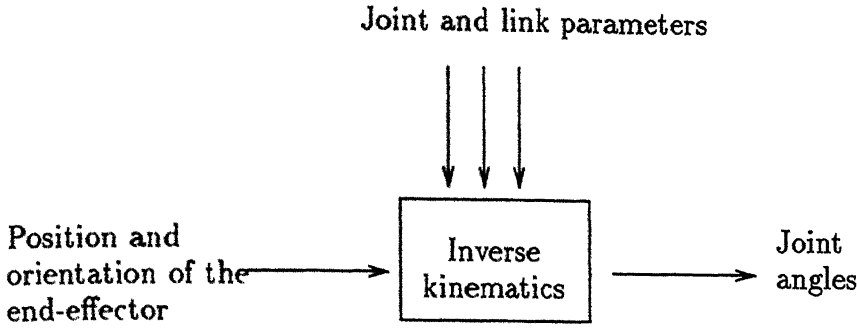


Figure 3.1: The inverse kinematics problem

for achieving the desired position and orientation of the end-effector. The inputs for this method are

- The present joint angles $\Theta = (\theta_1, \theta_2, \dots, \theta_n)^T$.
- The desired final transformation of the end effector 0T_n .
- The joint and link parameters.

This method outputs a set of joint angles which will yield the desired position and orientation of the end-effector. If such a set of angles do not exist this method fails.

3.2.1 Matrix equation of approximation

As a matter of convenience, we will write ${}^{i-1}A_i$ as A_i , where

$$A_i = \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

Now we have the matrix equation

$$A_1 A_2 \cdots A_n = T \quad (3.2)$$

where T is the desired transformation matrix for the end-effector. We analyze the articulated chain at a series of instantaneous positions. At any one of these positions if we have already achieved the desired transformation for the end-effector we stop the iteration process. Otherwise we go for one more iteration and refine the existing joint angles.

For the development of this iteration process it will be necessary to make some initial estimates of the values of the unknown joint angles. They may be expressed as

$$\theta_i = \bar{\theta}_i + d\theta_i, \quad i = 1, 2, \dots, n \quad (3.3)$$

Where $\bar{\theta}_i$ is the initial estimate and $d\theta_i$ is the error between the estimate and the exact value.

The matrix equation (3.2) may now be expressed as follows:

$$A_1(\bar{\theta}_1 + d\theta_1)A_2(\bar{\theta}_2 + d\theta_2)\cdots A_n(\bar{\theta}_n + d\theta_n) = T \quad (3.4)$$

When the transformation matrix (3.1) is written in terms of the initial estimates, it becomes

$$A_i(\bar{\theta}_i + d\theta_i) = \begin{bmatrix} \cos(\bar{\theta}_i + d\theta_i) & -\cos \alpha_i \sin(\bar{\theta}_i + d\theta_i) & \sin \alpha_i \sin(\bar{\theta}_i + d\theta_i) & a_i \cos(\bar{\theta}_i + d\theta_i) \\ \sin(\bar{\theta}_i + d\theta_i) & \cos \alpha_i \cos(\bar{\theta}_i + d\theta_i) & -\sin \alpha_i \cos(\bar{\theta}_i + d\theta_i) & a_i \sin(\bar{\theta}_i + d\theta_i) \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.5)$$

Using the trigonometric identities for sums of angles and assuming that the initial estimates are sufficiently accurate to make use of the small angle approximations on $d\theta_i$, the foregoing matrix can be expressed as the sum of two matrices.

$$A_i(\bar{\theta}_i + d\theta_i) = \begin{bmatrix} \cos \bar{\theta}_i & -\cos \alpha_i \sin \bar{\theta}_i & \sin \alpha_i \sin \bar{\theta}_i & a_i \cos \bar{\theta}_i \\ \sin \bar{\theta}_i & \cos \alpha_i \cos \bar{\theta}_i & -\sin \alpha_i \cos \bar{\theta}_i & a_i \sin \bar{\theta}_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} -\sin \bar{\theta}_i & -\cos \alpha_i \cos \bar{\theta}_i & \sin \alpha_i \cos \bar{\theta}_i & a_i \sin \bar{\theta}_i \\ \cos \bar{\theta}_i & -\cos \alpha_i \sin \bar{\theta}_i & \sin \alpha_i \sin \bar{\theta}_i & a_i \cos \bar{\theta}_i \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} d\theta_i \quad (3.6)$$

The first of these matrices can be identified as the original transformation matrix evaluated for $\bar{\theta}_i$. The second matrix, although not so easily recognized, can be seen to be the first partial derivative of the transformation matrix with respect to θ_i , also evaluated for $\bar{\theta}_i$. Henceforth the first of these two matrices will be written as \bar{A}_i . Thus

$$A_i(\theta_i) \cong \bar{A}_i + \left[\frac{\partial A_i(\theta_i)}{\partial \theta_i} \right]_{\theta_i=\bar{\theta}_i} d\theta_i \quad (3.7)$$

Noting that this problem is to be adapted to computer operation, a linear operator matrix, Q , will be introduced to perform the indicated differentiation through the following definition:

$$\frac{\partial A_i(\theta_i)}{\partial \theta_i} \bigg|_{\theta_i=\bar{\theta}_i} = Q \bar{A}_i \quad (3.8)$$

Under this definition the Q matrix is found to be

$$Q = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (3.9)$$

Condensing what has been shown thus far,

$$\begin{aligned} A_i(\theta_i) &\cong \bar{A}_i + Q\bar{A}_i d\theta_i \\ &\cong (I + Qd\theta_i)A_i \end{aligned} \quad (3.10)$$

and substituting these forms in equation (3.4) yields

$$(I + Qd\theta_1)\bar{A}_1(I + Qd\theta_2)\bar{A}_2 \cdots (I + Qd\theta_i)\bar{A}_i \cdots (I + Qd\theta_n)\bar{A}_n \cong T \quad (3.11)$$

Performance of the multiplication would yield a very lengthy and involved equation. However in keeping with the idea of the iteration process, all higher order terms of the form $(d\theta_i d\theta_j)$ may be neglected. The expanded form of this equation becomes

$$\begin{aligned} (Q\bar{A}_1\bar{A}_2 \cdots \bar{A}_n)d\theta_1 + (\bar{A}_1Q\bar{A}_2 \cdots \bar{A}_n)d\theta_2 + \cdots \\ \cdots + (\bar{A}_1\bar{A}_2 \cdots Q\bar{A}_n)d\theta_n = T - (\bar{A}_1\bar{A}_2\bar{A}_3 \cdots \bar{A}_n\bar{A}_n) \end{aligned} \quad (3.12)$$

Although this equation is still quite lengthy, and involves much matrix multiplication, it is symmetrical in form and can be handled nicely on a computer. This equation (3.12) will henceforth be called the matrix equation of approximation. It is the essential equation of the iteration process for, as will be shown, it may be solved for the $d\theta_i$ which are the iteration correction terms.

3.2.2 Derivation of simultaneous linear equations

Let the following notation be introduced to represent the terms of equation 3.12

$$B_0 = \bar{A}_1\bar{A}_2\bar{A}_3 \cdots \bar{A}_n \quad (3.13)$$

$$B_0 = \begin{bmatrix} B_{011} & B_{012} & B_{013} & B_{014} \\ B_{021} & B_{022} & B_{023} & B_{024} \\ B_{031} & B_{032} & B_{033} & B_{034} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.14)$$

$$\begin{aligned}
B_i &= (\bar{A}_1 \bar{A}_2 \cdots \bar{A}_{i-1} Q \bar{A}_i \cdots \bar{A}_{n-1} \bar{A}_n) d\theta_i \\
&= \begin{bmatrix} B_{i11} & B_{i12} & B_{i13} & B_{i14} \\ B_{i21} & B_{i22} & B_{i23} & B_{i24} \\ B_{i31} & B_{i32} & B_{i33} & B_{i34} \\ 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned} \tag{3.15}$$

With this new notation equation (3.12) becomes

$$B_1 + B_2 + \cdots + B_n \cong T - B_0 \tag{3.16}$$

Next define a single E -matrix to represent the entire left-hand side of this equation:

$$E_{jk} = \sum_{i=1}^n B_{ijk} d\theta_i \tag{3.17}$$

or

$$E = \begin{bmatrix} E_{i11} & E_{i12} & E_{i13} & E_{i14} \\ E_{i21} & E_{i22} & E_{i23} & E_{i24} \\ E_{i31} & E_{i32} & E_{i33} & E_{i34} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.18}$$

Finally the matrix equation of approximation reduces to the form

$$E \cong T - B_0 \tag{3.19}$$

Although simple to write, it should be remembered that this equation is only a shorthand notation for expressing the entire equation 3.12. It still involves $n-1$ unknowns, the $d\theta_i$, and must be solved for these to continue with the iteration process.

Equating both sides of equation (3.19) element for element would produce a system of twelve linear equations in n unknowns, the $d\theta_i$. To yield a unique set of solutions some of these equations must be eliminated as identities or as redundancies of other equations.

Note that the bottom row of both sides of equation (3.19) is null. Also for each of the matrix products B_i , all of the translational information (the a_i and d_i), is contained in the elements of the last column. Remembering that there are three translational degrees of freedom in space, it should be expected that there would be three equations representing this information. For this reason the matrix equation of approximation, equation (3.19) will be separated into its translational components

$$E_{14} \cong T_{14} - B_{014} \quad E_{24} \cong T_{24} - B_{024} \quad E_{34} \cong T_{34} - B_{034} \quad (3.20)$$

and its rotational components

$$\begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix} \cong \begin{bmatrix} T_{11} - B_{011} & T_{12} - B_{012} & T_{13} - B_{013} \\ T_{21} - B_{021} & T_{22} - B_{022} & T_{23} - B_{023} \\ T_{31} - B_{031} & T_{32} - B_{032} & T_{33} - B_{033} \end{bmatrix} \quad (3.21)$$

The matrix equation of approximation will be completely satisfied if and only if both of these conditions are satisfied. The first condition gives explicitly three of the required equations. Now we can extract the three equations representing the rotational information from the second condition, equation (3.21)

$$E_{21} \cong T_{21} - B_{021} \quad E_{31} \cong T_{31} - B_{031} \quad E_{32} \cong T_{32} - B_{032} \quad (3.22)$$

Up to this point all that has been done can be summarized in two steps.

1. The matrix equation of approximation was written as a function of the initial estimates, the link and joint parameters, and the errors involved, $d\theta_i$.
2. Six linear equations are isolated for the evaluation of these error terms.

Before proceeding with the solution of these six simultaneous linear equations, a few observations must be made about the convergence of the iteration process. With enough iterations, assuming that the initial estimates are sufficiently close, the system will converge so that the B_0 is the final matrix T . The only exception to this is at a dead point of the mechanism's operation. If the system were to converge on one of the forms which is not the unit matrix, the linkage would not be closed properly. For this reason it will be necessary to add the three diagonal elements as additional equations. This makes a total of nine simultaneous equations in n unknowns, the $d\theta_i$

3.2.3 Solving the simultaneous equations

Written in matrix form the simultaneous equations appear as follows:

$$\begin{bmatrix} B_{114} & B_{214} & \cdots & B_{n14} \\ B_{124} & B_{224} & \cdots & B_{n24} \\ B_{134} & B_{234} & \cdots & B_{n34} \\ B_{121} & B_{221} & \cdots & B_{n21} \\ B_{131} & B_{231} & \cdots & B_{n31} \\ B_{132} & B_{232} & \cdots & B_{n32} \\ B_{111} & B_{211} & \cdots & B_{n11} \\ B_{122} & B_{222} & \cdots & B_{n22} \\ B_{133} & B_{233} & \cdots & B_{n33} \end{bmatrix} \begin{bmatrix} d\theta_1 \\ d\theta_2 \\ \vdots \\ d\theta_n \end{bmatrix} \cong \begin{bmatrix} T_{114} - B_{014} \\ T_{124} - B_{024} \\ T_{134} - B_{034} \\ T_{121} - B_{021} \\ T_{131} - B_{031} \\ T_{132} - B_{032} \\ T_{111} - B_{011} \\ T_{122} - B_{022} \\ T_{133} - B_{033} \end{bmatrix} \quad (3.23)$$

On defining the M as the foregoing $[9 \times n]$ matrix of coefficients B_{ijk} , D as the column matrix of the unknowns $d\theta_i$, and V as the column matrix of constants B_{0jk} , equation (3.23) may be expressed as

$$MD = V \quad (3.24)$$

In general, this system of nine equations, $MD = V$ in n unknowns has no exact solution. However since the entire method revolves about an iterative approach, no significant error will be introduced if system is solved for closest approximation of a solution to all nine equations in the root-mean-square sense. All that is required to do this is to premultiply both the sides of equation (3.24) by the transpose of the M matrix.

$$M^T MD = M^T V \quad (3.25)$$

The matrix of coefficients is now square and, assuming that it is nonsingular, it may be inverted to give

$$D = (M^T M)^{-1} M^T V \quad (3.26)$$

This D -matrix gives explicitly the $n - 1$ unknown error terms, $d\theta_i$.

When the error terms $d\theta_i$ have been evaluated, they may be added to the initial estimates $\bar{\theta}_i$ to give an improved approximation of the exact values of the joint angle variables. It must be remembered that, by use of the small angle approximations, the $d\theta_i$ have been evaluated in radians. If greater accuracy is desired, these new approximations may be used as estimates $\bar{\theta}_i$, and the entire process may be repeated. The iteration process may be continued in this manner until the correction terms $d\theta_i$ are all smaller than the desired accuracy limits. At this point the mechanism has been completely solved for the final position of the end-effector.

3.3 Inverse kinematics in *Kalki*

In this we implemented the basic inverse kinematic routine as a function that takes the present angles, joint and link parameters, and the desired position of the end-limb as the inputs and outputs the joint angles that achieve the desired position and orientation of the end-limb.

Based on top of this function we can implement many applications that assist in the animation of the human body. For example one of the applications implemented is to fix a limb at a particular position in the world coordinates. This will be a great help to an animator when he moves the body but wants to keep a foot or palm at a particular position. This can be done by using the following formulations. First we find the world coordinate frame transformation for the foot at the present instant

$${}^{world}T_{foot_{t-1}} = {}^{world}T_{base_{t-1}} {}^{base}T_{foot_{t-1}} \quad (3.27)$$

Next we move the base located at the pelvis and find the desired transformation of the foot with respect to the base as follows:

$${}^{base}T_{foot_t} = {}^{base}T_{world_t} {}^{world}T_{foot_{t-1}} \quad (3.28)$$

Now we can use the inverse kinematic procedure for the portion from the base to the foot to derive the joint angles for the desired position and orientation.

Among other applications for inverse kinematics is the facility for specifying the path of the limbs during the motion. Using this we can calculate the position of the limb with respect to the base and evaluate the joint angles.

Chapter 4

System Design

The only way of finding the limits of the possible is by going beyond them into the impossible.

Arthur C. Clarke, British science fiction writer.

4.1 Introduction

In this we will consider the design philosophy and the motivating factors in the development of *Kalki*. *Kalki* is designed to be used by an animator who has little knowledge in computer science or mathematics. So, we tried hard to separate the technical details and the design aspects. All the major mathematical formulations like forward kinematics, inverse kinematics, stability, and dynamics are implemented as the basic building blocks on top of which many applications are developed. The design of *Kalki* is very flexible and open ended. It can be enhanced either for additional functionality or for efficiency. Since all the routines have well defined input-output relationship and clear functionality, we can plug-out a function and plug-in a new version without the slightest difficulty. For example the system currently uses the iteration method for inverse kinematics. Now if we want to implement a real-time application using this, we can replace this with a faster method like closed form solutions. For the present *Kalki* uses two types of bodies, a stick figure model and a polygonal model. we can add any number of additional models like spline based models as we wish. Similarly we can build a variety of backgrounds for animation. These backgrounds can be added to a library of backgrounds for *Kalki*. In the subsequent sections we will see in detail the design methodology of *Kalki*.

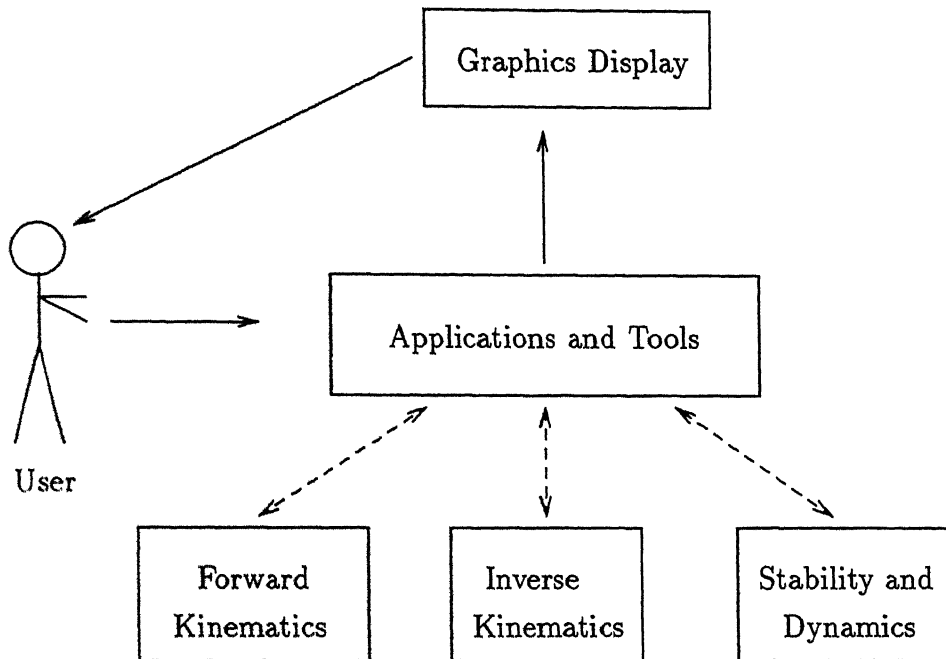


Figure 4.1: System structure in *Kalki*

4.2 Pose file hierarchy

In *Kalki* we introduced the concept of a pose file. These files are used to store body positions, and other information like camera positions and time distance between two body positions during animation. The human body currently implemented has 32 degrees of freedom. So, to store each key-frame we need to store 32 floating point numbers. The first type of files in the pose file hierarchy is the *dot-one files* (because these file names have the extension “.1”). These files store a number of key frames and the time distance between them. Built on top of this is pose file type called *dot-two files*. In these we combine a series of dot-one pose files. Associated with each dot-one file, we have a repetition factor which indicates the number of times the dot-one file has to be animated. Dot-two files also contain information about the camera position and orientation. The next level is the *dot-three files*. These are built on top of the dot-one files and facilitate the animation of multiple figures. These are made up of dot-two files and the corresponding character to be animated. While designing a dot-3 file, we can interactively specify the attributes of the character to be animated. For example we can specify the type of the body (stick figure, polygonal etc.), and other attributes like sex (male or female). This pose file hierarchy gives a comprehensive frame

work for animating multiple human bodies for any type of motion.

4.3 Pose file design

In *Kalki* we can interactively design the pose files. The user is provided with multiple views of the human body. We can change the posture of the body interactively. For this the user is provided with many tools which in turn use the basic routines like forward kinematics, inverse kinematics, and dynamics. The body can be placed anywhere in the world coordinate system using forward kinematics. The feet and other end-limbs can be fixed at the chosen positions and the body can be moved. After designing a key posture, we can store it in a pose file. It is frequently the case that most of the human body movements are periodic. So, *Kalki* provides a facility for generating the poses in successive periods. Also a single motion period is generally symmetric. For example in walking the period is symmetric with respect to the left and right limbs of the human body. So, After specifying half the period, *Kalki* can generate the other half simply by interchanging the left and right angles.

4.4 Pose file editing

Once a pose file is designed, we can do the after-design editing for getting the desired animation effect and for fine tuning the animation. We can load a pose file interactively. The system will ask for the number of the pose to be loaded, and then displays it in multiple views. now we can tweak with the posture and make adjustments. We can also vary the time distance between this pose and the previous one. This gives the effect of acceleration in the animation. Then we can store it back in the same or in a different file. Other facilities include inserting a pose in a pose file, deleting a pose from a pose file, and changing the camera positions.

4.5 Pose file animation

Kalki has a pose file animator that will take a pose file and produces the animation. In this we can interactively change the camera position to get a convenient view of the animation. We can also specify the body type to be used in the animation, and change the color settings. Before the animation starts, we can define a scaling factor for time. This gives us control

over the speed of the animation. Scaling factors less than one will speed up the animation and scaling factors greater than one will slow down the animation.

4.6 Stability checking

Kalki uses a built-in stability checking algorithm. Using this we can analyze the stability aspect of the human body. This routine will display the centers of gravity of the limbs and the whole body. Then it will find the support points on the ground and find the support polygon. With the help of this we can design stable positions. Also if the body is not found to be stable, then we can effects of a vertical collapse.

4.7 Scope for extensions

Since this project aimed to do so much in a short time, there are many things left for further extension. Specially the basic stability routine needs some applications to be built on top of that. Ideally given an unstable position, the system must be able to produce subsequent positions that will make the body regain stability. But this is a very difficult problem because of the regaining of stability depends on the future goal of the person. Since the computer can not predict what the future goal of the person is, it has to take some input from the user regarding this.

Implementation of the Lagrangian-Euler formulation for robot arm forward dynamics is one of the obvious extensions. Though there are many methods which are faster than this, it very much suits the formulations used for kinematics. In case a real time dynamics routine is needed, we can go for computationally more efficient methods. Once we have the basic dynamics routine, we can then design many applications for pose design. We can input the torques at each joint of the human body, and generate subsequent positions that result from the application of these torques.

The present system badly needs a face lift before it can be used commercially. A graphical user interface using X-Windows will be most suitable. In case we want to run the dynamics and inverse kinematics algorithms in real-time, we can run these routines on the fastest machine available and remote procedure calls (RPC) get the output on to the TSRX work stations.

4.8 Starbase implementation

The present system is implemented on the HP-TSRX 3D-Graphics work station. The graphics routines should be used very cautiously if we want to make the system portable. In *Kalki*, all the basic mathematical formulations are coded as distinct functions without any graphics routines in them. So, to port the current system on to some other machine, we only have to change the few graphics routines in the system.

Starbase is a powerful graphics package provided by the Hewlett-Packard company. Some of the features available are

- 3D primitives.
- 3D camera viewing.
- Hidden surface removal.
- Light sources.
- Shading and texture.
- Double buffering.

In *Kalki* each limb is modeled as a quadrilateral mesh. We are using hidden surface removal and shading for realistic animation. Conventional animation used in films is very simple in principle. We show the successive frames of the film very fast (i.e., 30 frames per second). In computer graphics we have many ways of doing the animation. These are

Color table animation. In this we plot the moving body at different positions using different color table entries. Now we make only one color visible and all the rest of the color table entries are set to the background color. We traverse the color table sequentially and make only that color visible. This provides a method of doing very fast animation. The disadvantage is that we can not do general purpose animation with this.

Sprite animation. In most of the video games and PC based games, a small object moves against a fixed background. This can be done easily by first storing the small portion of the background where we are going to draw the object. We then draw the object.

Next time the object moves, we first restore the stored background and repeat the process.

General 3D animation. In this use the same technique used in conventional animation.

This process is made very smooth by using a technique called double-buffering. In this the frame-buffer is divided into two parts. At a time only one buffer is enabled for displaying on the terminal. In the mean time we draw the next frame in the other buffer and then we switch the buffers.

For some reason¹ starbase uses left-handed coordinate system. So, initially the left-handed coordinate system of the world is transformed to the right-handed coordinate system of the base of the human body. We use the modeling transformation matrix of the starbase to draw each limb in its own coordinate system. In kalki we tried to minimize the number of functions using graphics function calls. This facilitates easy porting. Two functions `draw_body()` and `draw_background()` are the main graphics routines. For enhancing the speed of the animation these two routines must be optimized.

¹ Known only to the designers of starbase.

Chapter 5

Using Kalki

The time has come, the walrus said,
to talk of many things.

L. carrol, Alice in wonder land.

5.1 Introduction

In this chapter we will consider in detail how to use the human body animation system *Kalki* . Though the animator is relieved of the burden of knowing the underlying mathematical formulations, he still needs to know some concepts about graphics and the issues of starbase. For example, starbase uses the concept of a camera for viewing the objects in the world. Hence the the animator has to know about the setting of camera in starbase. This chapter also discusses some design issues and mentions those parts of the code that have to be modified for enhancing the functionality of *Kalki* . *Kalki* has five options in the main menu. They are

- Design a dot-1 pose file.
- Design a dot-2 pose file.
- Design a dot-3 pose file.
- Animate a dot-3 pose file.
- Edit a dot-1 pose file.
- Quit *Kalki* .

Now we will consider each of these options in the following sections.

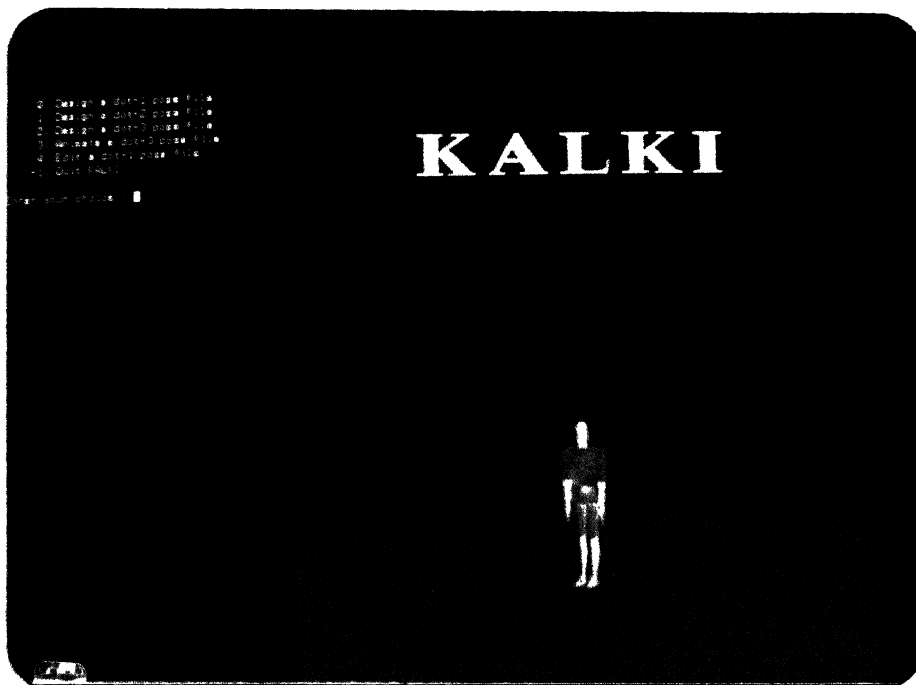


Figure 5.1: Main menu in *Kalki* .

5.2 Design a dot-1 file

When the animator selects this option he is provided with a set of tools for designing a dot-1 pose file. He is provided with the following options.

- Change front view camera.
- Change side view camera.
- Change oblique view camera.
- Change side view2 camera.
- Change Kalki's pose.
- Save the pose.
- Load a pose.
- Initialize Kalki.
- Use inverse kinematics.

- Fix left foot.
- Fix right foot.
- Release left foot.
- Release right foot.
- Quit pose design.

He is also provided with four views of a human body. The human body is initialized such that all the joint angles are set to zero. He is initially positioned on a carpet of 10×10 meters dimension. The initial position of the base coordinate frame at the upper part of the pelvis is 500 centimeters along X-axis, 500 centimeters along Z-axis, and, 109 centimeters along the Y-axis. The height of the pelvis is the sum of the lengths of the foot, shin, thigh, and the pelvis. These lengths are defined as constants in the header file `define.h`. By changing these values we can simulate bodies with varying dimensions. In each viewport two light sources are simulated for lighting effects. Specular reflections are turned on for better realism.

Now we will consider each of these tools and how to use them.

5.2.1 Changing the cameras

The first four options in the pose design menu are for changing the cameras for the four viewports shown. Initially the cameras are set such that we get one front view, two side views, and one oblique view. However, the animator can change the cameras according to his choice. When any of these four options are selected, we are shown the current camera setting, and are prompted to enter the new camera setting. Each camera has twelve parameters. The three parameters *camx*, *camy*, *camz* specify the position of the camera in the world coordinate frame. One more triple *refx*, *refy*, *refz* specify the reference point at which the camera is focused. Three more parameters *upx*, *upy*, *upz* are used to specify the orientation of the camera. Two parameters *front*, *back* are used to specify the clipping planes from the reference point. The parameter *field_of_view* specifies the angular scope of the camera. Finally *projection* specifies whether parallel or perspective projection is used.

When the user changes any of the camera settings, he can see the result immediately in the corresponding viewport, and he can independently change the camera settings for the

four viewports shown. One common need is that arises is that, we want to see whether the feet are touching the ground or not. For this we have to set both *camx*, and, *refx* to zero.

5.2.2 Change Kalki's pose

We select this option for changing the posture of the human body interactively. The user is provided with a menu that indicates the different joint angles, and other degrees of freedom available to the user. There are totally 32 options available. The last six options are for changing the position and orientation of the whole body. The base coordinate frame of the body is fixed at the top of the pelvis. This is approximately the place where the naval of the body is. Each limb is listed along with the degrees of freedom available for that limb. We use the notation F/B standing for forward-backward movement. L/R stands for left-right movement. ROT stands for rotating along with it's axis. When the user selects a specific degree of freedom, he is provided with the current value of that degree of freedom, and he is prompted to enter the new value for that degree of freedom. Immediately the changes are made visible in the four views shown to the user. Similarly the user can modify the pose until he gets the desired pose. The last six options need special mention. Starbase uses left-handed coordinate frame. So, the Z-axis is going into the screen. For example if we want to bring the body closer to the user, we have to decrease the value of Z.

5.2.3 Save the Pose

This option is used for saving the pose into a dot-1 file. When this option is selected the user is prompted for the file name. He is then asked to specify the camera for viewing. This camera setting can be done until we get the desired viewing position. We also have to specify the number of frames that have to be generated between this frame, and the previous frame. This option opens the dot-1 file in append mode. So, the new pose is appended to the existing poses.

5.2.4 Load a pose

This option is used for loading a specific pose into the system. This prompts for the dot-one file from which the pose has to be loaded. Then it will tell the user how many poses are there in that dot-one file, and asks the user which pose has to be loaded. The user then enters the pose number that has to be loaded. After this, that pose is made the current

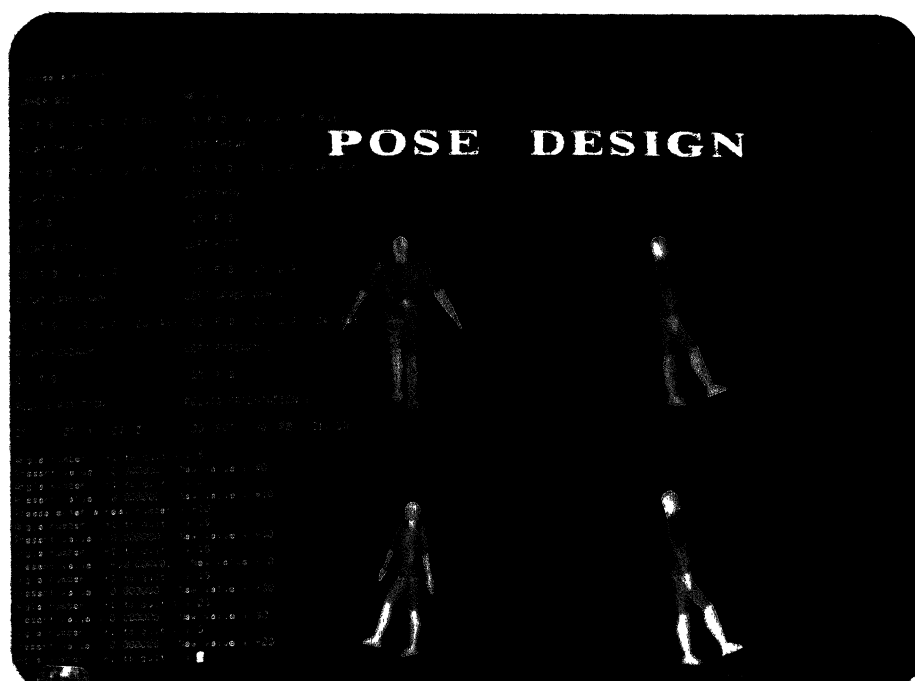


Figure 5.2: Changing body posture in *Kalki* .

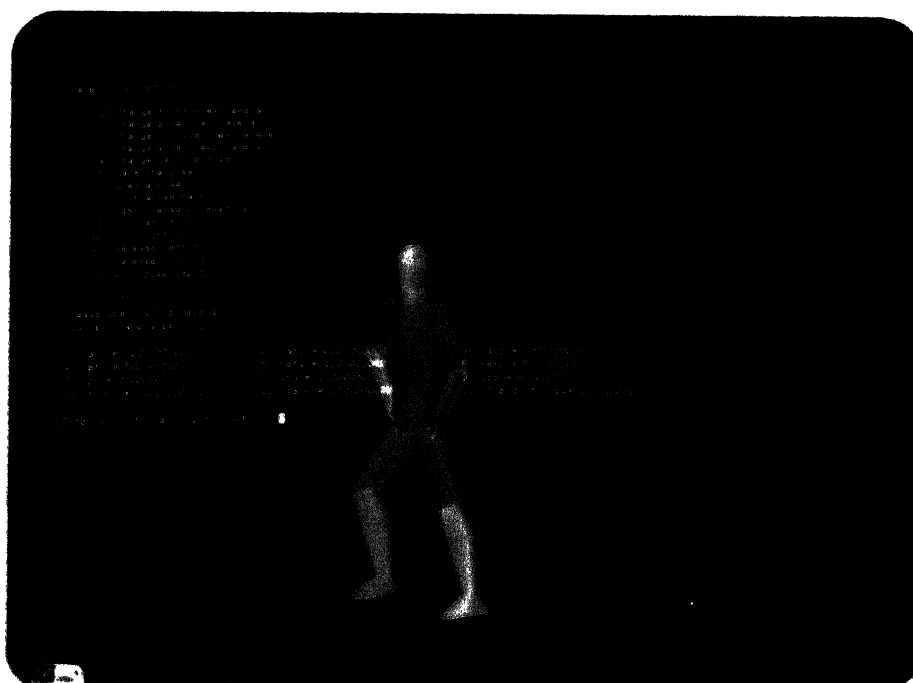


Figure 5.3: Specifying camera for dot-1 file.

pose of the human body. The corresponding human figure is displayed in the four views. Now the user can modify this pose, and save it to any of the dot-one file.

5.2.5 Initialize kalki

Some times it so happens, that many angles of the body are changed, and we don't like that pose. Now before start designing afresh, we have to reset all the angles to zero. When the user selects this option, the system will do this. Note that the system will not change the position of the body.

5.2.6 Use inverse kinematics

This is a tool that is very useful when designing human walking or any other motion, where we need to fix certain limbs at some world coordinate position. For example when a person walks, one foot is fixed on the ground, while the other foot travels in the air. In this case it is vary useful if the system can calculate the angles automatically for the leg that is fixed on the ground. When the user selects this option, he is provided with seven further options. They are

- Fix left foot.
- Fix right foot.
- Fix left palm.
- Fix right palm.
- Fix neck.
- Move base.
- Apply inverse kinematics.

The first four options are used for fixing the respective limbs in the world coordinate frame. The system will find out the current position of the limbs, and stores it in the memory. Now using the option six, we can change the position of the body in the world coordinate frame. When this option is selected, we are given the full menu for changing any degree of freedom in the body. However it doesn't make sense, if we change the joint angles of that limb which has been fixed.

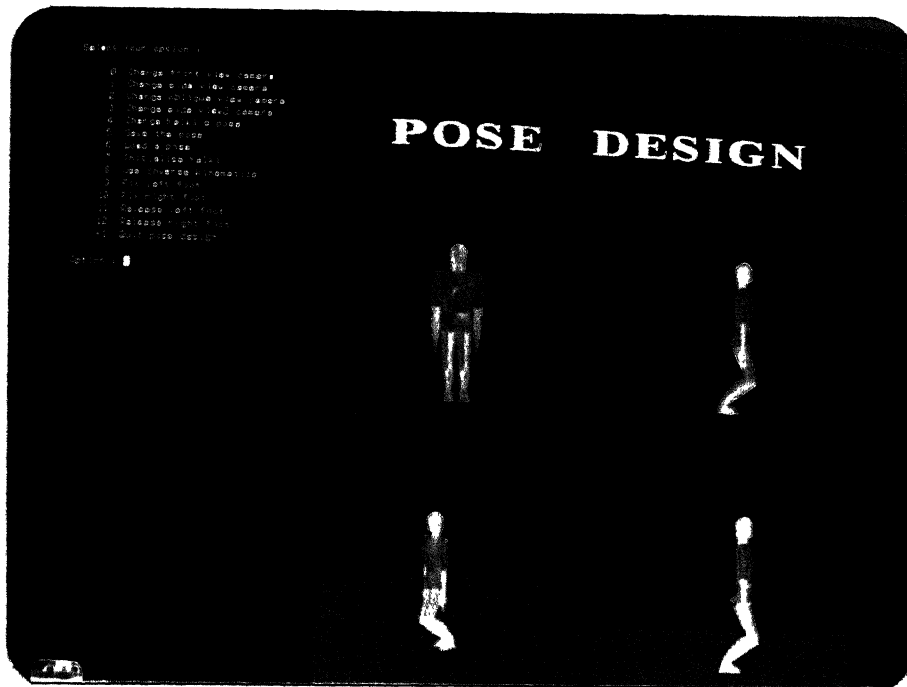


Figure 5.4: Using inverse kinematics in *Kalki* .

After this, the user selects option 6, if he wants the inverse kinematics to be applied. Now the system will find the joint angles, such that the final position of the limb is the same as that stored before. Some times it may so happen that we move the body by a large distance or we move it to an impossible position. In this case the inverse kinematics algorithm may not be able to find the solution. If this happens, the system will inform the user about this, and returns to the menu.

5.2.7 Fix the limbs

This option looks similar to the one in inverse kinematics. But it is different in the sense that there is no calculation of joint angles in this case. When the user selects any of these two options we find the world coordinate positions of the feet. When the user modifies an angle, we calculate the difference between the new position and the old position of the feet. We then move the base coordinate frame by this distance to get the feet to their original position. These two options are extremely useful, because otherwise the user has to bring back the body by trail and error.

When we select these two options, they remain active until we specifically deactivate

them using options 12, and 13.

5.3 Design a Dot-2 pose file

Once we have designed the basic dot-one files, we can go for the design of dot-two files. Dot-two files are meant for combining several dot-one files. Each dot-one file has a repetition number associated with it. If the repetition number is zero, that dot-one file is animated only once. If the repetition number is n , then the corresponding dot-one file is animated $n + 1$ times. The idea behind this dot-two file is as follows. When we design any sequence of motion, we have some periodic as well as nonperiodic motions. Let us consider one particular example of human walking. Initially the body is at rest. Then it moves to the starting of the one stride period. After that we have periodic strides. Finally the body comes back to the rest position. This situation is best modeled by a dot-two file with two dot-one files. The first dot-one file will contain the motion from the rest position to the starting of the stride with repetition number zero. The second dot-one file will contain the motion of one walking stride, with repetition number equal to the number of strides we want. The third dot-one file will contain the motion from the end of one walking stride to the rest position, with the repetition number zero. When we have a repetition number greater than zero, the body position is automatically adjusted after each period by the animation routine, such that the next period starts from the final position of the first one.

This option prompts the user for the name of the dot-two file. It then asks him to enter a series of dot-one files and corresponding repetition numbers. This dot-two file can easily be edited using an ordinary text editor.

5.4 Design a dot-3 pose file

Once we have defined all the necessary motions, we have to define the character that should perform the animation. A dot-3 file contains a sequence of characters and corresponding dot-2 files. Each character is animated using the motion specification in the corresponding dot-2 file. When this option is selected, first the user is prompted to enter the name of the dot-3 file. Then, he is asked to define a series of characters with corresponding dot-2 files. Presently a character has the following attributes.

- Name.

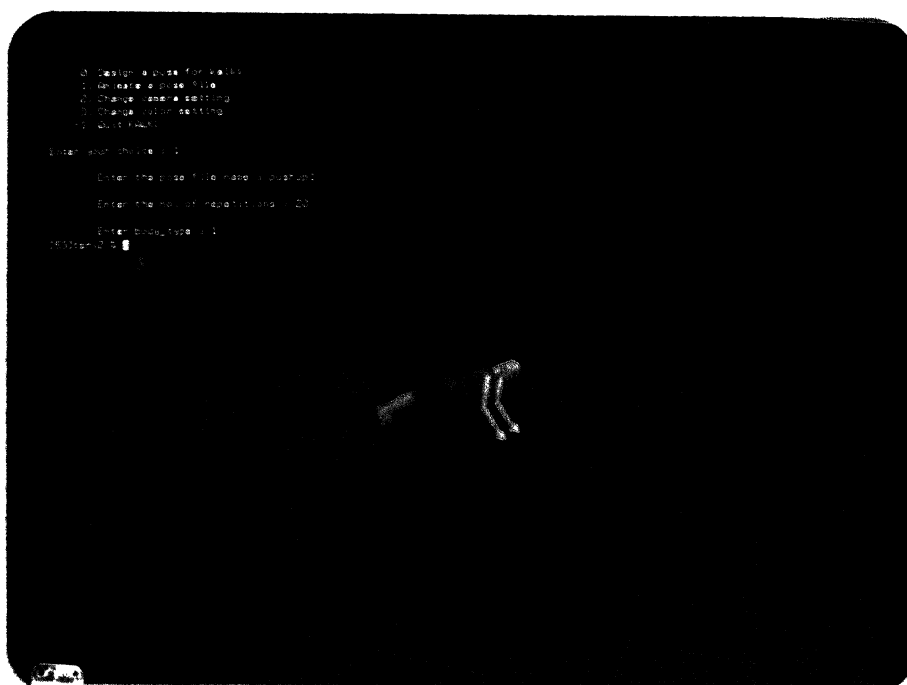


Figure 5.5: Doing pushups in *Kalki*

- Sex.
- Body type. (stick figure, or polygonal)
- Colors of limbs. (there are 14 colorable limbs in the body)

Once we have defined a dot-3 file, we can animate and see the result using the next option.

5.5 Animate a dot-3 pose file

This option enables the user to animate a dot-3 file. It asks for two things. First it asks for the name of the dot-3 file. It then asks for the name of the *hero* in the animation. As we have already mentioned, there can be many characters in the animation. Each character has an associated dot-2 file. Now since we have the camera specification for each character in the dot-1 files, there is some ambiguity about which character's camera is to be used. For this, we use the concept of a hero. When we enter the name of a hero, the system will automatically follow the camera of that character whose name is the same as that of the hero.



Figure 5.6: Multiple character animation in *Kalki*



Figure 5.7: Dancing in progress.

Before the animation starts, the system will first open the dot-3 file, and the dot-2 file specified in it, and all the dot-1 files specified in each of the dot-2 files. It will load all the necessary information into a main memory data structure. Then it starts the animation. The definitions of the data structures used are found in the file `define.h`. So, if we want to add further information in the pose files, these definitions are to be changed.

A routine, called `drawbody()`, is used for drawing the human body. This takes a structure `BODY` as a parameter. In this structure we have all the attributes defined for a character. The routine `drawbody()` in turn calls `draw_limb()` for drawing each limb. If we want to change the degrees of freedom in the body, the file `forwardk.c` should be modified.

In each frame, before drawing the body, we draw the background. Right now the background is simply a 10×10 meters carpet. The background is drawn by a routine called `draw_background()`. This routine has no parameters. But if we want to animate the background, we have to pass some parameters to it.

5.6 Edit a dot-1 pose file

First the user designs the pose file. then, he animates and sees the result. If he doesn't like it, then he can edit the dot-1 files using this option. We should note that most of the information regarding the animation is present in the dot-1 files. Editing of dot-2, and dot-3 files can be done using a text editor.

In dot-1 file editing the user is first asked to enter the name of the dot-1 file to be edited. Then he is provided with the following options.

- Edit a new dot1 file.
- Save.
- Save to a new file.
- Translate poses.
- Translate cameras.
- Edit a record.
- Insert a pose.

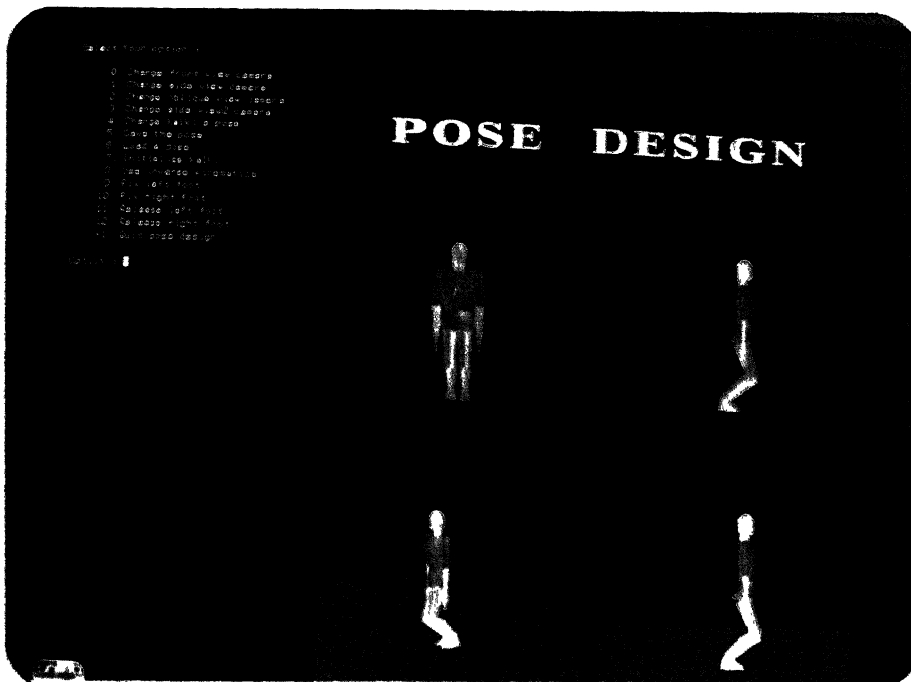


Figure 5.8: Editing a pose in *Kalki*

- Delete a pose.
- Quit dot1 editing.

Now discuss each of the above options.

5.6.1 Edit a new dot-1 file

This and the next two options are related to loading and saving the dot-1 files. When we load a dot-1 file, we are shown the first frame in the file. After making the changes we can save the changes using the second option. The third option is used for saving the changes to a new dot-1 file.

5.6.2 Translate poses

This is a very useful option for designing multiple character animation. Using this we can translate all the poses to a new place in the world coordinate system. A typical application of this is described below. Say, we want to design a dance sequence involving three characters. Now we define a dot-1 file for each step in the dance. Each step can be repeated any

number of times. Now we have a sequence for one character. Now we edit the dot-1 files and translate them to different places in the world for getting the sequences for the other two characters. Note that after translating the poses, we have to save the poses to a new dot-1 file.

5.6.3 Translate cameras

This option is similar to the above one, but this translates all the camera positions specified. When this option is selected the user is prompted to enter the increments for all the camera parameters. These increments are added to all the camera positions in the dot-1 file. Unless we want to change the camera settings of the original file, we should save this to a new file.

5.6.4 Edit a record.

This is used for editing a particular record in the dot-1 file. The user is first told the number of poses present in the file. Then he is asked to enter the pose number to be edited. This pose is then shown on the screen, and the user is provided with the following options.

- Change pose.
- Change camera.
- Change distance.

Using the above options we can change the pose of the human body, the camera position, or the frame distance between the current frame and the next frame. Note that changing the distance can be used to simulate accelerations.